APD test board V.2 operation

Leon Mualem --University of Minnesota
8/27/2005 http://www.hep.umn.edu/~mualem/oa/apd_v2/APD_V2.doc

Conceptual design

The APD test board uses a Fermilab MASDA[1] ASIC to integrate AC coupled signals from a Hamamatsu 32 pixel APD array model S8550. The MASDA-X chip performs dual correlated (before/after) sampling of the inputs. There are 2 sets of sample capacitors for each channel, referred to as A and B. Only one set is used in the prototype board since the 2 sets will have slightly different offsets and slightly different gains. The MASDA has 128 channels, only 32 of which are used and readout. The Xilinx CPLD supplies integrator reset, integration timing, readout logic, and readout clock timing signals as multiples of the externally supplied clock, which runs at 1MHz nominally. The digitized data are sent out for every trigger and every channel on the data interface lines.

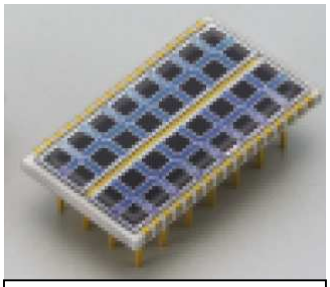


Figure 1
Hamamatsu S8550

APD readout board design

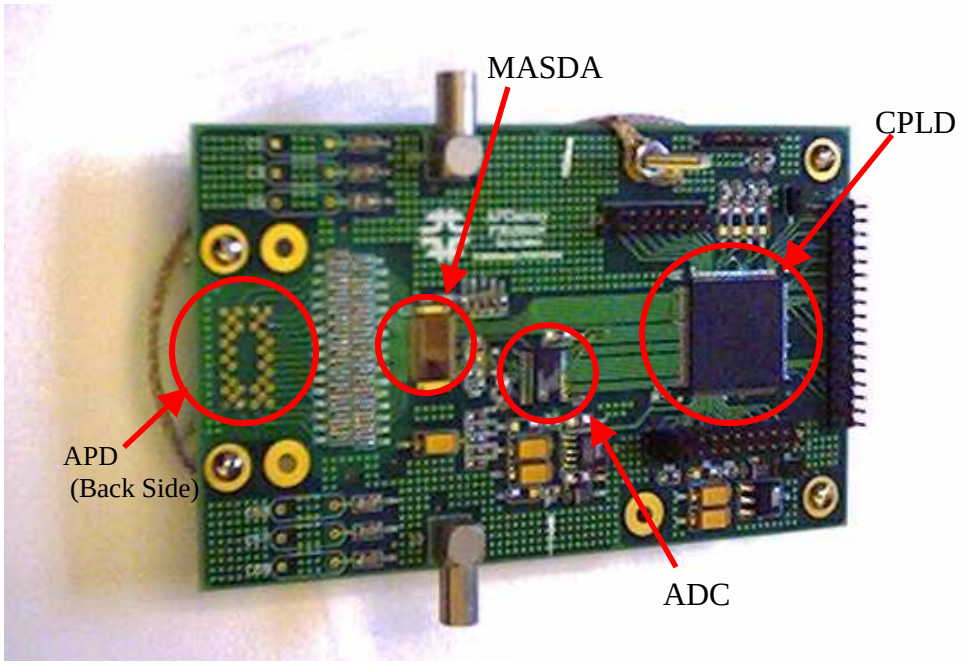


Figure 2 APD readout board version 2 with major components indicated

In production the MASDA is covered to protect the delicate wire bonds underneath. High voltage (~400V) for the APD arrays are supplied via the lemo connectors. The voltage for the version 2 board described and shown here is **POSITIVE**. There are 2

independent diodes in each S8550 package which may require different bias voltage to run at the intended gain. The power connector J1, the upper LEMO connector in Figure 2, supplies HV power to the even channels, 0-30, and J7 supplies power to the odd channels, 1-31. In addition Figure 2 shows the layout of the board and the major components, the APD itself is mounted on the back side, the MASDA, the ADC, and the CPLD. On the right hand side of the figure is the main interface connector, a 34 pin header. This is used to supply the clock signal (1MHz TTL compatible), power (+5V) and a trigger signal as inputs. The output data bits are also on this connector as well as a signal indicating the start of the integration time. All the pins at the edge of the board are connected to the ground plane. The remaining 17 pins are arranged as in Table 0.1.
The clock and power signals are electrically connected to 2 pin headers J2 and J6 respectively. This provides the flexibility of driving the board with signals from the interface cable or separately.

Signal Line	Signal
1	←Clock (1MHz TTL) →
3	←POR
5	TRIG→
7	Data Valid→
9	D0→
11	D1→
13	D2→
15	D3→
17	D4→
19	D5→
21	D6→
23	D7→
25	D8→
27	D9→
29	SBEF→
31	Spare
33	←Power (+5V)→
2,4,6,…,32,34	GND

Table 0.1J4 header signals

Additional interface connectors

There are several additional interface connectors on the board that can be programmed to provide additional configuration control or output signals if needed by simply reprogramming the CPLD. The programming connector is TS1, the single row header located at the top of the board in Figure 2, pin 1 is located at the left hand side. Programming can be done with a Windows PC with a PC Parallel Port and a programming cable from Xilinx (costs about $100). The programming takes only seconds using free programming software from Xilinx. The connector pinout is arranged as indicated in Table 0.2. Headers J3 and J5 can be programmed as configuration jumpers or additional outputs if needed. They are programmed as in Table 0.3 and Table 0.4.

JTAG Pin	JTAG signal
1	TCK
2	TDI
3	TDO
4	TMS
5	VCC
6	GND

Table 0.2 TS1 header signals

The signals on the J3 connector are output signals for monitoring board operation.

J3 Pin	Signal
1	ABSEL
3	SBEF
5	RC2
7	RC3
9	COMB_CLK_UNUSED
11	COMB_TRIG
13	COMB_POR
15	COMB_CLK
2,4,…14,16	GND

Table 0.3 J3 signals (the upper/smaller one)

The signals on the J5 header are both output and inputs to the CPLD. The first 3 signals are again for monitoring of the board operation. The other signals are inputs to change the operation of the MASDA if needed. The BW_INP_X pins are used to control the bandwidth (risetime) of the integator output. The default is the fastest, at 360-500ns. Jumpering these pins will decrease the bandwidth, and increase the risetime. The GAIN_INP_X signals can be used to change the gain of the MASDA. The default is the second highest gain. Currently jumpering the pins will lower the gain, by adding binary weighted capacitors to the integrator feedback. The first will make the gain about 1/3, the second about 1/5, and the third about 1/9. The other signals, EXT1 and 2 are used to control the phase of readout for the MASDA, since we only use ¼ of the channels it has to skip them on readout, and these pins are used to adjust it in case the bonding is different for different boards somehow. (Should come properly jumpered, with only EXT1 jumpered. Don't remove it.)

J5 Pin	Signal	Input/Output
1	ACQ	Output
3	READY	Output
5	DONE	Output
7	NC	--
9	BW_INP_3	Input
11	BW_INP_2	Input
13	BW_INP_1	Input
15	GAIN_INP_3	Input
17	GAIN_INP_2	Input
19	GAIN_INP_1	Input
21	EXT2	Input
23	EXT1	Input
2,4,…,22,24	GND	--

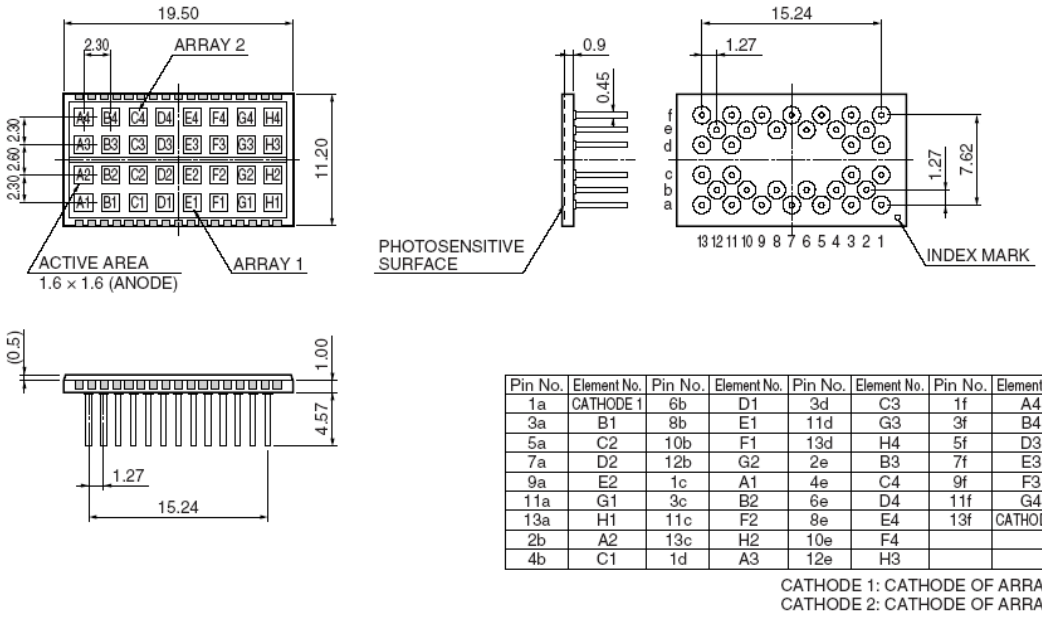
Table 0.4 J5 signals (the bigger/lower one)

APD references and description

The Hamamatsu S8550[2] APD is a package of 2 electrically independent APD arrays of 2x8 pixels. The pixel size is 1.6mm square and the pixel pitch is 2.3mm in both directions. The spacing between the two arrays is 2.6mm to allow for the ceramic insulator between the 2 arrays. The array is arranged as a common cathode configuration and requires a bias voltage of approximately 400V. The actual specification for each APD will come from Hamamatsu and from tests during the checkout procedure for the board with the mounted APD.

The back side of the APD has been potted with RTV to inhibit discharge which may happen if there is some contamination or moisture allowed to form. This has happened in practice, and the RTV should prevent it from happening in the future. The result is the death of at least the one diode array that sparks.

■ Dimensional outline (unit: mm)



Pin No.	Element No.	Pin No.	Element No.	Pin No.	Element No.	Pin No.	Element No.
1a	CATHODE 1	6b	D1	3d	C3	1f	A4
3a	B1	8b	E1	11d	G3	3f	B4
5a	C2	10b	F1	13d	H4	5f	D3
7a	D2	12b	G2	2e	B3	7f	E3
9a	E2	1c	A1	4e	C4	9f	F3
11a	G1	3c	B2	6e	D4	11f	G4
13a	H1	11c	F2	8e	E4	13f	CATHODE 2
2b	A2	13c	H2	10e	F4		
4b	C1	1d	A3	12e	H3		

CATHODE 1: CATHODE OF ARRAY 1
CATHODE 2: CATHODE OF ARRAY 2

Figure 3 Pixel arrangement and dimensions from Hamamatsu Specifications Sheet for Si APD S8550.

Pixel Number	Readout Channel	Pixel Number	Readout Channel
A1	0	A3	1
A2	2	A4	3
B1	4	B3	5
B2	6	B4	7
C1	8	C3	9
C2	10	C4	11
D1	12	D3	13
D2	14	D4	15

E1	16	E3	17
E2	18	E4	19
F1	20	F3	21
F2	24	F4	23
G1	22	G3	25
G2	26	G4	27
H1	28	H3	29
H2	30	H4	31

Table 0.5

Mechanical Design/Layout

For anyone interested in physical interface to the board the dimensions and hole sizes etc. for mounting are found in the gerber files that are posted here:
http://www.hep.umn.edu/~mualem/oa/gerbers/APD2-13-04PM.zip
The 8 large mounting holes near the APD and at the other end are 0.125", and are located relative to the lower left corner of the board (oriented so you can read the designer's name.) are as follows:

X location (mils)	Y location (mils)
200	800
200	2200
600	800
600	2200
3250	250
3250	2750
4750	250
4750	2750

Table 0.6 Large mounting hole locations, the first 4 are near the APD for mounting the cooler, and the next four are at the other end of the board for mounting to a box.

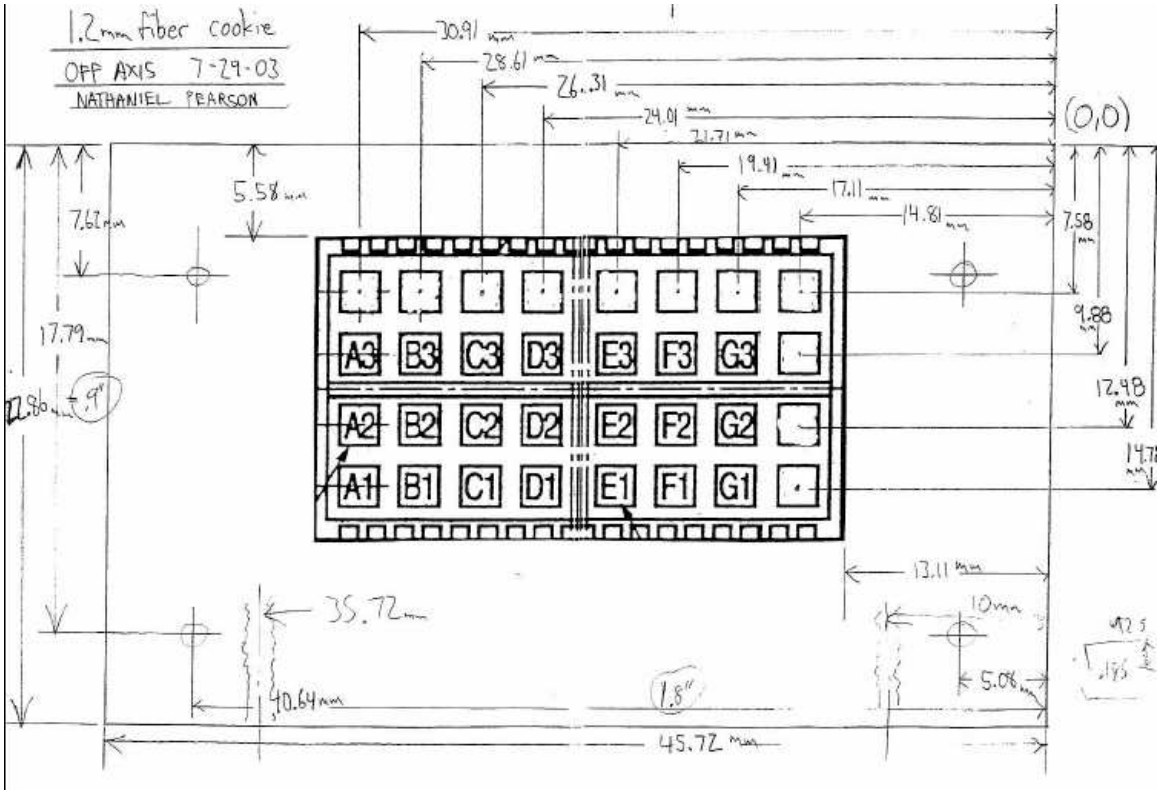


Figure 4 Layout sketch of 1.2mm fiber cookie with dimension in mm.

Board operation

The readout board has two operational states, an acquire mode and a readout mode. The acquire mode clocks the IRST, SBEF, SAFT, and ABSEL signals to the MASDA. It first resets the integrator to VREF, then it holds the before sample on a capacitor on the falling edge of SBEF, this starts the integration time. The after sample is then held when SAFT goes low ending the integration time and the acquisition of this sample. The integral is then the difference between the before and after samples. This is dual-correlated sampling. If the board did not receive a trigger (rising edge on J4 pin 3) before SAFT goes low another acquire sequence will be initiated. This will continue until a trigger is received and the CPLD switches to readout mode.

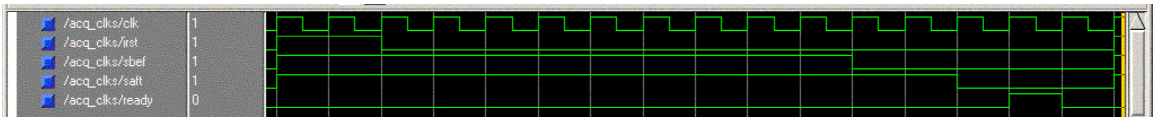


Figure 5 MASDA acquire clock timing as programmed.

IRST is held for 2 clock cycles, when the minimum recommended is 1us. The 2 clock cycle setting is conservative, in order to minimize noise. The long time after IRST goes low before SBEF goes low is to allow more time for the sample capacitors to settle. The bandwidth is halved since the SBEF and SAFT capacitors are both connected during this time. The minimum recommended is about 3 us in this configuration. The 9 clock cycles is again conservative in order to minimize the noise. The next 2 clock cycles after SBEF

goes low are the actual integration time. Since the risetime of the output stage of the integrator is about 500ns the actual live time is less than 2us. In order to fully collect the signal the signal window (that should be used for coincidence triggering) starts when SBEF goes low and last 500ns. The SBEF signal is on the interface connector for just this purpose. If the coincidence window is longer the amount of charge collected will depend on the actual time of the signal, so the variation will be greater.

In readout mode the acquired signals are multiplexed onto the output bus of the MASDA where they are digitized by the 10-bit ADC three times. The CPLD adds the data from these samples together to average the signal. This averaging reduces the noise by about 5% and increases the effective LSB size, or increases the number of bits to almost 12. Each sample takes 16us to read out, and the 12 bits are read out as 2 6 bit words so that the entire board can be read out in little more than 1ms. The data are sent out asynchronously to the PC, so the board will function even without a PC reading the data.

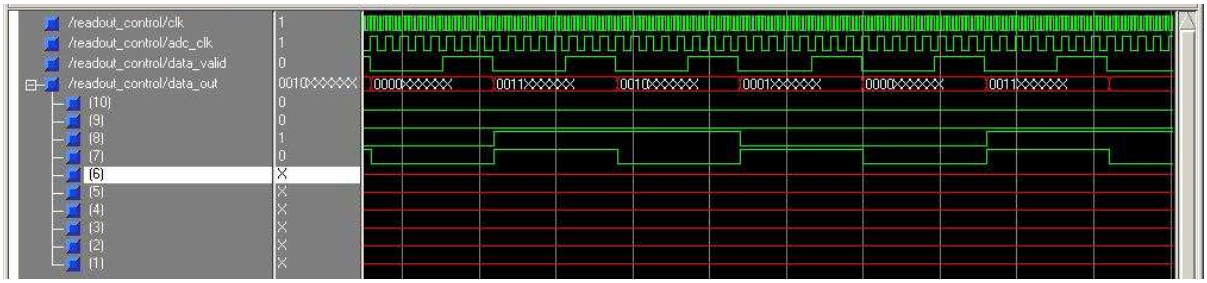


Figure 6 Readout clock sequence showing 2 full data samples.

Data acquisition programs

There are two data acquisition programs that have been tested to work with the board. The first is programmed in NI LabWindows and provides a GUI to drive the board, select the number of samples, output files etc. The limitation of this program is that it must be run with one of the older consumer versions of windows, 95/98/Me. This is due to restrictions against direct port access in NT based systems. This slows down the access too much and the data acquisition may become unreliable. It might work for some systems and not for others, therefore I recommend using a computer with the previous operating systems for reliable operation. There is also a program written in C++ that can interface with root and run under Linux. At the moment it can acquire data correctly, but does not have a nice interface. It also requires that you be able to perform direct port reads and disable interrupts, so it must at least be SUID root. (If you don't understand this, go with the windows.)

The port that is used for data acquisition is a standard parallel port set to bi-directional (BYTE) mode, which allows reading of 8 bits at a time. It should work with just about any parallel port setting, but is somewhat more likely to work if it is set to ECP in the BIOS. This enables the program to set the mode to BYTE correctly. The speed of the computer is essentially irrelevant. I have had it working reliably on 200MHz Pentium up to 800MHz P-III with essentially no difference in acquire speed. (Since the parallel port is so slow, it would probably work on a PC-XT.)

In order to acquire data the PC takes control of the TRIG line on the interface connector with the parallel port –STROBE line. This allows the PC to control when acquisition may occur. This line can either be directly connected to the TRIG line, or broken out to use in triggering logic; this mode will be covered later. The PC polls the DataValid line from the DAQ board. This line is connected to pin 12 of the parallel port, the -PE (printer error) bit, and is read by reading the status byte of the printer port, which is usually 0x379. The data bits 0-7 from the DAQ board are connected to data bits 0-7 on the parallel port cable. Data bits 8 and 9 from the DAQ board go to –ACK and –Busy respectively. The bits 8 and 9 are vestigial, and could be used if needed, but do not currently convey any information. The two 6 bit data words are sent on the bits 0-5 lines, and bits 6 and 7 indicate the phase of the data: Bit 6 indicates whether the 6 bits are Most or Least significant bits, and Bit 7 indicates whether it is the Q or I mode channel of the ADC that digitized the signal. The DataValid line indicates, as expected that the data on the data bus is valid. This signal goes high one clock cycle after the data are presented, which is more than enough time for the bits to settle. The entire readout cycle is simply watching DV go high and low and recording the data. Since the DAQ program knows what to expect next it is able to detect a missed sample. When this happens the entire set is simply discarded and a new acquire/readout cycle is started. This means you only get complete and correct information. If anything is amiss during a sample it is discarded.

One set of 6 bits is read in a minimum of 3 reads of the port. The first is when the polling sees that -DV has gone low the second is when it reads the 8 bits from the port, and the third is another read to the –DV bit to ensure that it is still low, and that the data did not become invalid during our read. A typical port read time is 1.2-1.6us under Linux or the consumer windows versions (98 or ME). With the data persisting for 15us there is plenty of time to perform these three reads. Under NT the typical port read times are 4-6us, which does not allow enough time to perform all 3 reads reliably. It is possible to lengthen the DV time, but then you run into another problem on WinNT, that you can not disable interrupts and may be interrupted during your readout cycle which leads to a different sort of unreliability. It may be possible to provide the slow readout as a jumper selection, but it would probably be a last resort, and it would better if everyone simply uses a “reliable” OS.

LabWindows APDdaq

I intend to keep a recent and functioning copy of the DAQ software in http://www.hep.umn.edu/~mualem/oa/apd_v2/apddaq/.

Data acquisition is controlled by a simple program called apddaq.exe. The control panel for this is shown in Figure 7. The buttons on the left hand column provide some diagnostics. They are vestigial, and will probably be removed at some point, though they may provide some useful diagnostics if there are problems. The “Quit” button on the bottom is functional, and will quit the program at any time. The selector in the middle, “Board type” selects between different hardware versions of the APD board. It defaults

to “V2 avg”, which is appropriate for most users, as there is only one V1 board in existence. The selection is available so that the program will be identical in all locations to minimize debugging problems. The “Samples to acquire” control sets the number of DAQ cycles that will be attempted. On each cycle all 64 channels will be read. A cycle is only counted if all channels read out successfully. If not, the program will discard any acquired data and try to again to get a valid set of samples. The “Go” button starts the data acquisition, which will continue until reaching the selected number of samples. The “STOP” button will stop the data acquisition essentially immediate. The “Output File” control sets the name of a file that the acquired data will be dumped into. The target file is emptied on each data acquisition if it exists, if it does not exist it will be created and filled. Clicking on the textbox will open a dialog to choose a file, from an explorer type navigation box, or simply type the file name. For use with the analysis programs, it might be useful to take some pedestal data as a “.ped” file. If you take light injection, and name the pedestal file filename.ped, and the light injection file as filename.li, it can be very useful.

The controls near the bottom right corner serve as progress indicators. The “Samples acquired” progress bar will grow as samples are acquired, and the text box will show the actual number of samples acquired in steps determined by the “Increment” control. If you are reading noise or light injection, an increment of 100 is appropriate. The DAQ proceeds at about 100Hz, so it is easy to tell that it is still alive, and does not consume a lot of resources in updating the display. If you are acquiring cosmic ray data, which will come in much slower (depending on size of detector, triggering you choose, etc.) it might be appropriate to set the increment all the way down to 1.

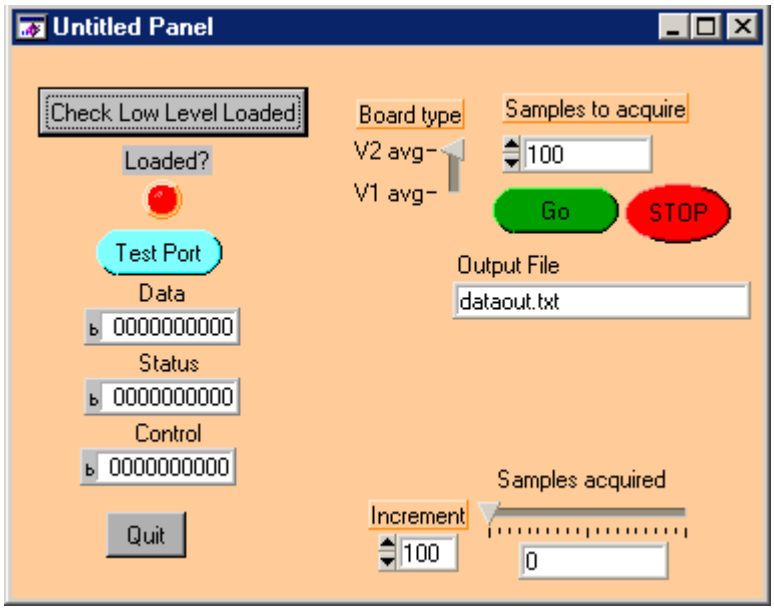


Figure 7 LabWindows DAQ panel.

When the program starts a couple of status lines will be printed in a stdout window. This might include a warning about non-ECP modes which is probably innocuous, and it will

report the time to 1 million port reads. This number should be about 1-2us/read. If it is substantially longer, there is something rather strange, and DAQ may be unreliable.

Upon completion of the desired number of samples a report will be printed to stdout. This will contain the mean and RMS for each channel, and the average RMS and RMS of the RMS values for each entire array. There are 2 arrays on the board, one reads out on the even channels, and the other the odd channels. These are reported separately since it is possible that you might run one array and not the other with the proper bias voltage.

Linux APDdaq

Currently out of date, so unavailable. Let me know if this is a deeply desired feature.

DAQ Hardware

The current DAQ hardware consists of one cable that connects to the APD board on one end and the PC parallel port on the other through an "RS-232 jumper box"[3]. The jumper box really has nothing to do with RS-232, it is simply a configurable DB-25 male to DB-25 female adapter where all connections are made via jumper wires. This is how the APD board signals are connected to the appropriate parallel port signals. The connections are as shown in Table 0.7.

Signal Line	APD Board Signal	Parallel Port Signal	Parallel Port Pin
1	←Clock (1MHz TTL) →		
3	←POR		
5	←TRIG	–STROBE	1
7	Data Valid→	PE	12
9	D0→	D0	2
11	D1→	D1	3
13	D2→	D2	4
15	D3→	D3	5
17	D4→	D4	6
19	D5→	D5	7
21	D6→	D6	8
23	D7→	D7	9
25	D8→	–ACK	10
27	D9→	–BUSY	11
29	SBEF→		
31	Spare		
33	←Power (+5V)→		
2,4,6,…,32,34	GND	GND	18-25

Table 0.7 APD readout board signals and their connections to parallel port lines.

External Triggering

In this connector the –STROBE line is actually not directly connected, the TRIG line and –STROBE signals actually come out on LEMO connectors. If they are joined with a LEMO barrel then the board will switch to readout mode and send the data out after completion of the next acquire sequence that completes after TRIG goes high. If they are disconnected then it can be used for external coincidence triggering. The way this works is to use the –STROBE TTL signal from the PC to indicate that the PC is ready to read out the data. The other part of an external trigger is determined by the SBEF signal on the data cable. The integration window starts when SBEF goes low. The total integration time is 2 clock periods. Due to the risetime of the output of the integrator in the MASDA it takes 1.5us to collect the entire charge. Therefore the coincidence gate for a valid signal should be the AND of the –STROBE line and a gate pulse that can be generated when SBEF goes low and lasts $2*T_{clock}$-1.5us. For example if your clock is running at 1MHz T_{clock} is 1us, so the gate is 0.5us long starting when SBEF goes low. If there is a coincidence, then the external logic should generate a high TTL level on the TRIG line to indicate there was a valid coincidence trigger and that the data should be read out. This signal must come before 4 clock periods have elapsed from the time SBEF goes low. This is typically 3.5us from the integration gate, which is quite a long time for typical logic modules, so should not be a problem. If there was not a TRIG high signal the APD board will start another acquire sequence immediately.

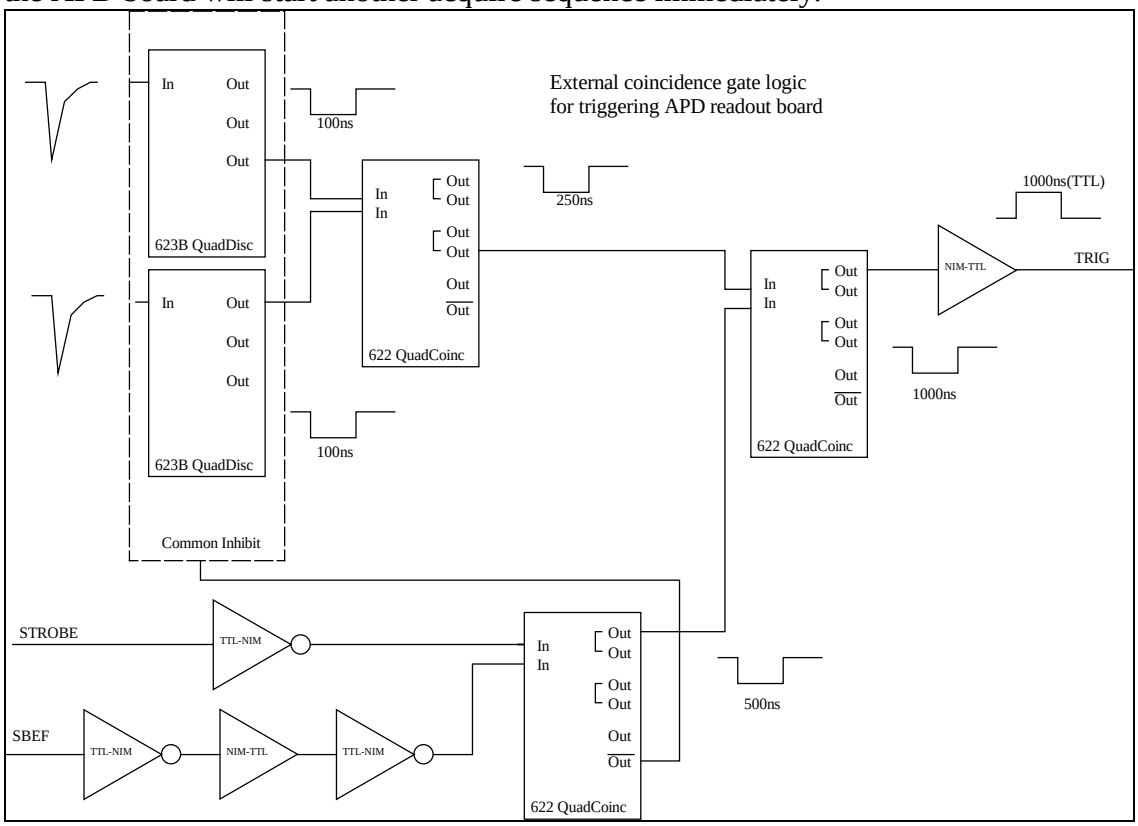


Figure 8 Trigger logic used for external coincidence logic. Gate widths are approximate. Modules shown are all standard LeCroy NIM modules, the buffers and inverters are all LRS688 level translators. The triple buffering of SBEF is necessary to debounce a somewhat noisy signal, only one may be necessary.

The typical way generate the gates and perform this logical comparison is by using standard NIM logic modules. It is particularly useful to have gate generators, discriminators (for external trigger counters), coincidence generators, and TTL-NIM level translators for interfacing the two standards. These can be loaned from FNAL PREP. The scheme that I used is shown in Figure 8.

CPLD program

Updated CPLD information can be found at http://www.hep.umn.edu/~mualem/oa/apd_v2. This contains a directory call CPLD_APR13 that contains the actual programs/components used to build the programming file for the CPLD. There are hundreds of files here, as this is a copy of the entire directory. Most of the files are of no real use to you, but there are a few interesting ones. The most interesting ones being ACQ_CLKS.vhd, which show the states/timing for the acquire sequence. Also, readout_control.vhd shows the sequence for the digitization and readout sequence. There is also a postscript of the figure with the program schematic, which might print more legibly as top.sch.ps.

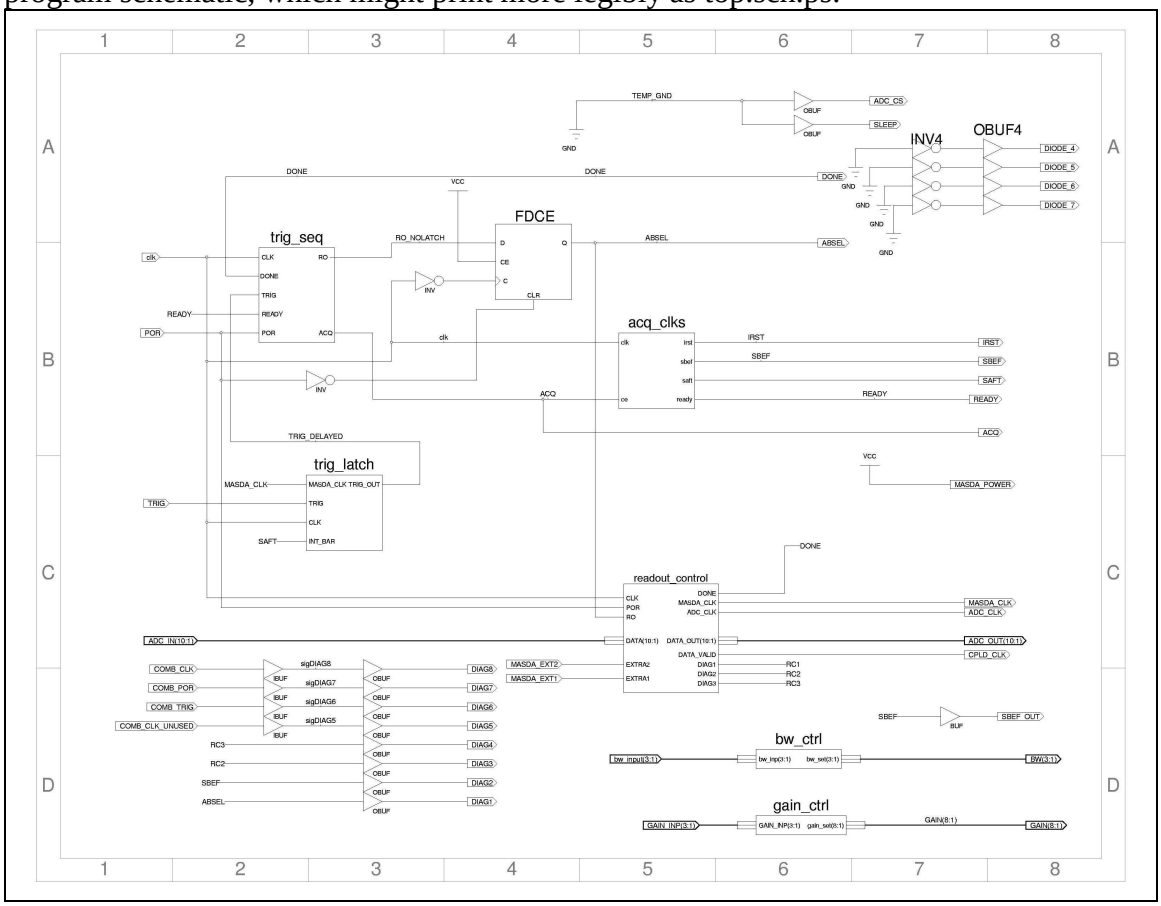


Figure 9 Schematic of operation of Xilinx CPLD.

Analysis programs

I have written a few Root macros that are useful for analyzing the data files. They should be found at http://www.hep.umn.edu/~mualem/oa/apd_v2/macros/.

The data file format is simply channel number and ADC value for all 32 channels for each acquired sample. This makes it rather simple to write your own analysis program in whatever language and using whatever OS you want. I chose to use Root, so that it would be portable and not require a compiler. (You can use ACLIC on them if you want.) The first program is called pedonly.C, (you might need pedonly.h also for ACLIC) The program will take a single filename and make an ntuple of the data, compute pedestals for each channel, and fill an ntuple with pedestal and common mode noise subtracted data. The common mode subtraction reduces the noise about 10%. It should be a valid operation since in general the occupancy of the detector will be very low, so it is reasonable to use some fraction of the lowest data on each array to do common mode subtraction, and thus reduce the noise. Although the common mode noise is correlated between the two arrays, it is calculated separately for each array in case only one is operating in a given file.

Running the program on a pedestal data file called mydata.ped would look like this:
root –l 'pedonly.C("mydata.ped")'
This will produce the two ntuples mentioned above using the data from mydata.ped, and two histograms, one for each array, of the pedestal and common mode noise subtracted data. It will also fit the histograms with a Gaussian which will indicate the noise level as sigma, reported in stdout, and plotted on the histograms.

There is another program for displaying light injection data. It works best if you take pedestal and light injection data with the same filename with .ped and .li extensions respectively. First you have to load the pedonly.C file as above, then run the liwped.C file with the filename with no extension as the argument, e.g. : .x ./liwped.C("filename"). That will analyze the file filename.ped to find the pedestal values to subtract from the data in filename.li containing the light injection data. It will finish displaying a lego plot of the light injection pulse height distributions for all 32 channels.

The program apddisplay.C can be used to display the hit pattern and event by event pulse heights in the apd array. An example picture of a track in a 4x8 cell scintillator tracker is shown in Figure 10. The color of the box and the text indicate the pulse height detected in units of photoelectrons assuming APD gain of 100 and the standard readout board conversion gain of 208e-/mV.

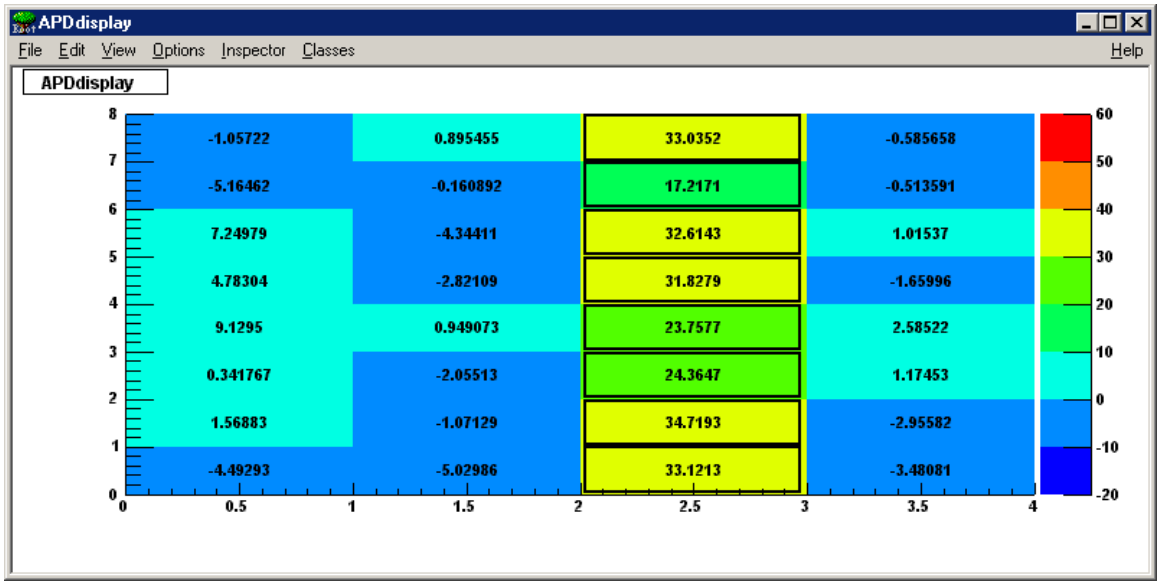


Figure 10 Sample track in a 4x8 array of scintillator strips arranged as a tracking module.

---

[1] MASDA reference information available on web at http://library.fnal.gov/archive/test-tm/2000/fermilab-tm-2063.pdf. Additional references with more or less the same information are here:
NIM A 485 (2002) 661-675. "Design and performance of a low noise 128 channel ASIC preamplifier for readout of active matrix flat panel imaging arrays"
and
T. Zimmerman, The MASDA-X chip—a new multi-channel ASIC for readout of pixelated amorphous silicon arrays, Fermilab technical note FERMILAB - TM-2063, 1998.
and
R. Yarema, et. al., "A Programmable, Low Noise, Multichannel ASIC For Readout of Pixelated Amorphous Silicon Arrays," presented at the 8th European Symposium of Radiation Detectors June 14-17, 1998, S&loss Elmau, Germany. Submitted to NIM.

[2] Details of the Array are found here http://usa.hamamatsu.com/assets/pdf/parts_S/S8550.pdf

[3] This is Digi-key Part number AE1039-ND.